Synopsys Design Constraint — язык задания временных ограничений на примере Altera TimeQuest. Часть 3

Денис ШЕХАЛЕВ
shdv@micran.ru

В предыдущих частях статьи мы рассмотрели основы временного анализа и научились задавать временные ограничения тактовых частот различных проектов. Но рассмотренные примеры не содержали задание временных ограничений для интерфейсов ввода/вывода, что представляет собой наибольшие сложности при разработке sdc-файла.

Правильное задание временных ограничений интерфейсов ввода/вывода вызывает наибольшие сложности у начинающих разработчиков для ПЛИС и пользователей TimeQuest. А владение навыком устранения временных нарушений в интерфейсах представляет собой своего рода «черную магию» TimeQuest. Но в этой части статьи вы увидите, что в этом нет ничего сверхсложного.

Но сначала автор хочет сделать небольшое отступление. Все термины, введенные в этой части статьи,— авторские и по своему значению могут не совпадать с терминами, которые приводятся в документации фирмы Altera. Это сделано для того, чтобы внести однозначность в рассмотрение различных интерфейсов и не путать читателя. Начнем с азов.

Виды интерфейсов ввода/вывода

Существует великое множество интерфейсов, но все их можно разделить на две основные группы: синхронные и асинхронные. Это разделение основано на методе обработки сигналов этих интерфейсов в ПЛИС. Если для обработки сигналов интерфейса используется логика, тактируемая от частоты этого интерфейса, то такой интерфейс является синхронным. В противном случае интерфейс асинхронный.

Синхронные интерфейсы разделяются по признаку местоположения источника тактовой частоты интерфейса на System Synchronous и Source Synchronous. В свою очередь, асинхронные интерфейсы подразделяются на асинхронные с обработкой на комбинационной логике и асинхронные с обработкой на системной тактовой частоте.

Есть еще самосинхронные интерфейсы, работающие с Clock Data Recovery (CDR), но их мы рассматривать не будем. Также не будем рассматривать подробно реализацию интерфейсов, нас интересуют только вопросы задания временных ограничений в TimeQuest.

Синхронные интерфейсы

Начнем с рассмотрения синхронных интерфейсов, так как методы временного анализа, реализованные в TimeQuest, ориентированы прежде всего на них. Синхронные интерфейсы разделяются на два вида по месту происхождения источника тактовой частоты интерфейсного устройства:

- System Synchronous — это интерфейсы (рис. 21), в которых тактовая частота идет непосредственно с ПЛИС на интерфейсное устройство. К таким интерфейсам можно отнести АЦП/ЦАП, память, шину в режиме Master и т. д.
- Source Synchronous — это интерфейсы (рис. 22), в которых тактовая частота идет от отдельного генератора (в качестве которого может выступать само интерфейсное устройство) к ПЛИС. К таким интерфейсам можно отнести шину процессора, к которому ПЛИС подключена как Slave, АЦП/ЦАП и т. д.

Перед тем как углубиться в детали, еще раз напомним основные положения временного анализа, которые нам потребуются:

1. Временной анализ синхронных схем (в том числе интерфейсов) рассматривает передачу данных от регистров, тактируемых частотой источника, до регистров, тактируемых частотой приемника.
2. TimeQuest при анализе синхронных интерфейсов использует только ту информацию о тактовых частотах, которую ему указали. Если одна из тактовых частот не указана, то винить нужно прежде всего разработчика, а не TimeQuest.
3. Для выходных интерфейсов ПЛИС нужно помнить, что у регистра-приемника есть два временных параметра: ***Tsetup (tsu)*** — время предустановки данных (время, в течение которого данные должны быть неизменны *до* фронта тактовой частоты) и ***Thold (th)*** — время удержания данных (время, в течение которого данные должны быть неизменны

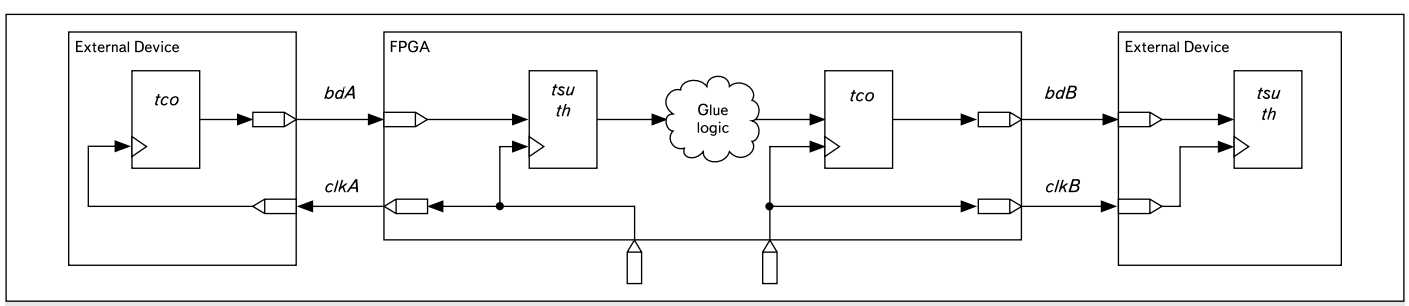


Рис. 21. Структурная схема системы System Synchronous

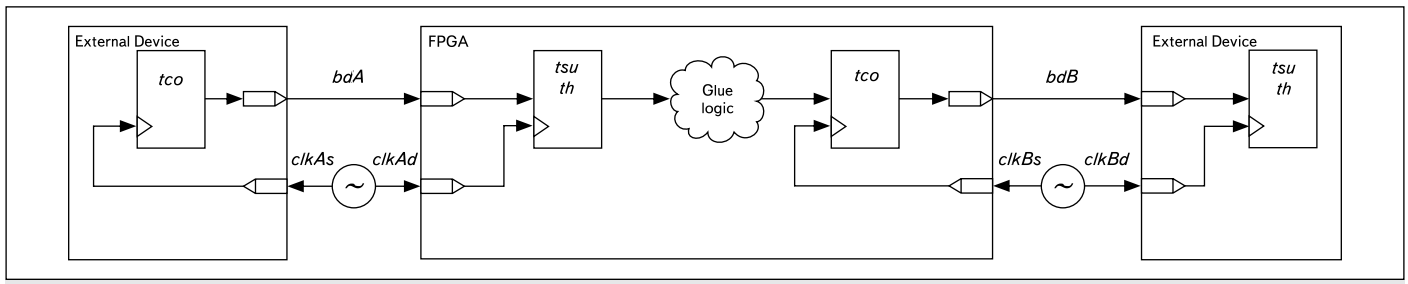


Рис. 22. Структурная схема системы Source Synchronous

после фронта тактовой частоты). Если эти времена нарушаются, то возможны сбои в работе регистра. Эти сбои называются метастабильностью.

4. Для входных интерфейсов ПЛИС нужно помнить, что в зависимости от типа устройства (регистр/АЦП/память/и т. д.) у регистра-источника могут быть следующие важные параметры. Для регистров это ***Tclock-to-out (tco)*** — время появления данных на выходе после фронта тактовой частоты. Для памяти это ***Tacces*** — время появления данных на выходе после фронта тактовой частоты и ***Thold*** — время удержания данных на выходе после фронта тактовой частоты.

System Synchronous Output

Рассмотрим вывод пилы на синхронный параллельный ЦАП:

```
module dac (input iclk, output oclk, output logic [7 : 0] data);

  logic [7 : 0] cnt;

  always_ff @(posedge iclk) begin
    cnt <= cnt + 1'b1;
    data <= cnt;
  end

  assign oclk = iclk;

endmodule
```

Положим параметры ЦАП ***tsu/th = 5ns/5ns***, частота работы — 10 МГц, все настройки проекта в Quartus по умолчанию. Файл задания временных ограничений для этого проекта будет таким:

```
set_time_format -unit ns -decimal_places 3
derive_clock_uncertainty

create_clock -period 10MHz -name {iclk} [get_ports {iclk}]
create_generated_clock -name {oclk} -source [get_ports {iclk}] [get_ports {oclk}]

set_clock_groups -exclusive -group {iclk oclk}

set_output_delay -clock [get_clocks {oclk}] -max 5.0 [get_ports {data[*]}]
set_output_delay -clock [get_clocks {oclk}] -min -5.0 [get_ports {data[*]}]
```

Рассмотрим строки sdc-файла, которые до этого нам не встречались:

```
set_time_format -unit ns -decimal_places 3
```

Здесь мы задаем наносекунды в качестве единиц времени и устанавливаем точность задания времени в три десятичных знака.

```
create_generated_clock -name {oclk} -source [get_ports {iclk}] [get_ports {oclk}]
```

Это описание сигнала тактовой частоты, на котором работает приемник нашего интерфейса (ЦАП). Судя по коду модуля, никаких преобразований с сигналом тактовой частоты не было, поэтому он описывается один в один. Так как в проекте есть синхронная передача данных между частотами ***iclk*** и ***oclk***, они помещены в одну логическую группу.

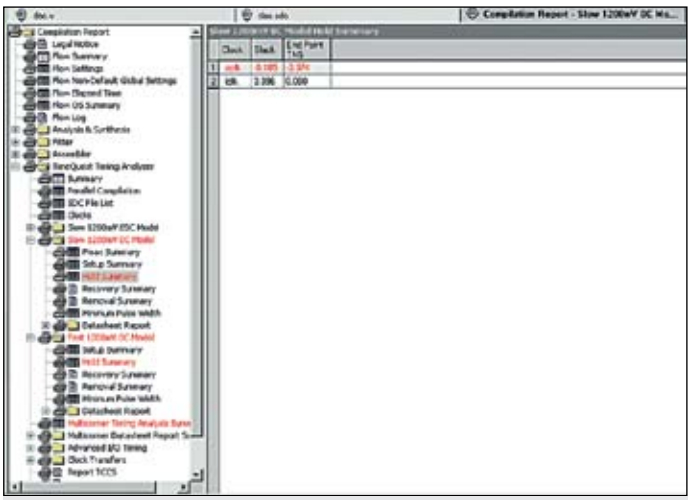


Рис. 23. Отчет Quartus о сборке проекта dac

```
set_output_delay -clock [get_clocks {oclk}] -max 5.0 [get_ports {data[*]}]
set_output_delay -clock [get_clocks {oclk}] -min -5.0 [get_ports {data[*]}]
```

Это задание временных ограничений интерфейса по ***tsu = 5ns*** и ***th = 5ns*** ЦАП соответственно. Задание ***tsu/th*** происходит относительно сигнала тактовой частоты, который подается на ЦАП. В примере используются непосредственные значения параметров ЦАП, но в общем случае формула задания параметров для таких интерфейсов следующая:

```
Output maximum delay value = maximum trace delay for data + tSU of external register – minimum trace delay for clock
Output minimum delay = minimum trace delay for data – tH of external register – maximum trace delay for clock
```

В этом примере мы пренебрегаем задержками распространения сигналов на плате, предположим, что пути сигналов данных и тактовой частоты выровнены. Все, мы задали временные ограничения для нашего проекта. Собираем в Quartus, запускаем анализ и видим нарушение временных ограничений в отчете (рис. 23). Казалось бы, частота всего

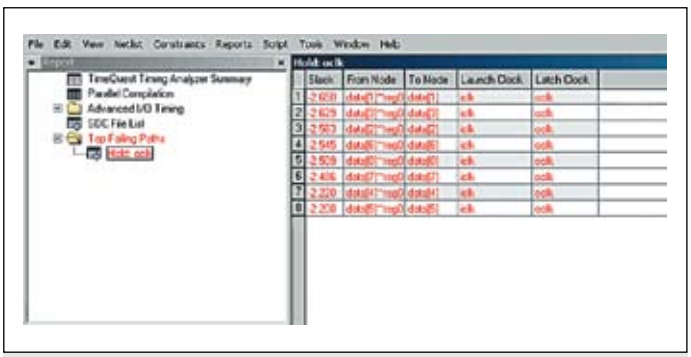


Рис. 24. Нарушения временных ограничений в проекте dac

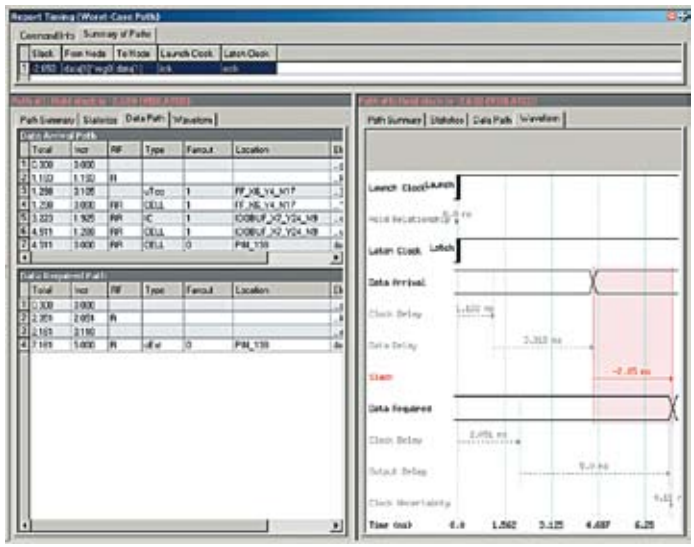


Рис. 25. Подробный отчет о нарушении временных ограничений в проекте dac

10 МГц, какие могут быть нарушения? Давайте разбираться. Запускаем TimeQuest, ставим режим анализа ***MIN_fast_1200mv_0c***, выполняем анализ и видим результат (рис. 24).

Да, действительно не укладываемся в заданные временные ограничения. Для того чтобы разобраться в причине нарушения, с помощью команды ***Report Worst Case Path*** смотрим, что именно происходит в интерфейсе (рис. 25).

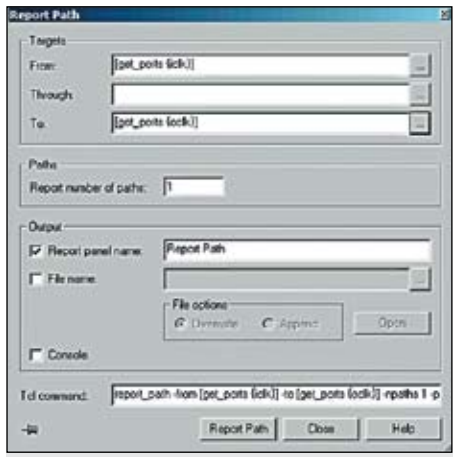


Рис. 26. Диалоговое окно для временного анализа конкретного пути

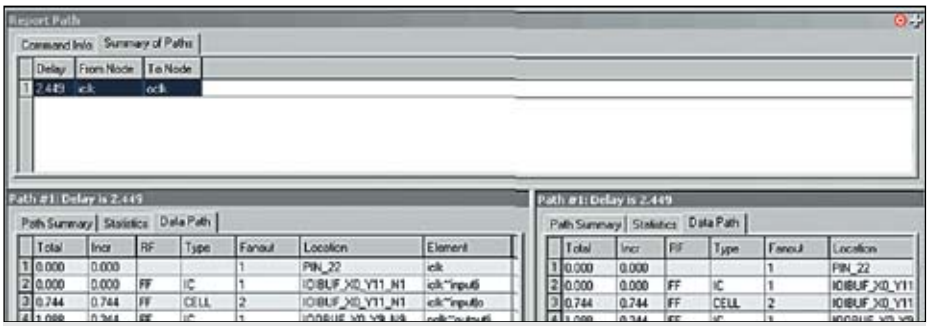


Рис. 27. Результат анализа задержки пути между сигналами iclk и oclk проекта dac

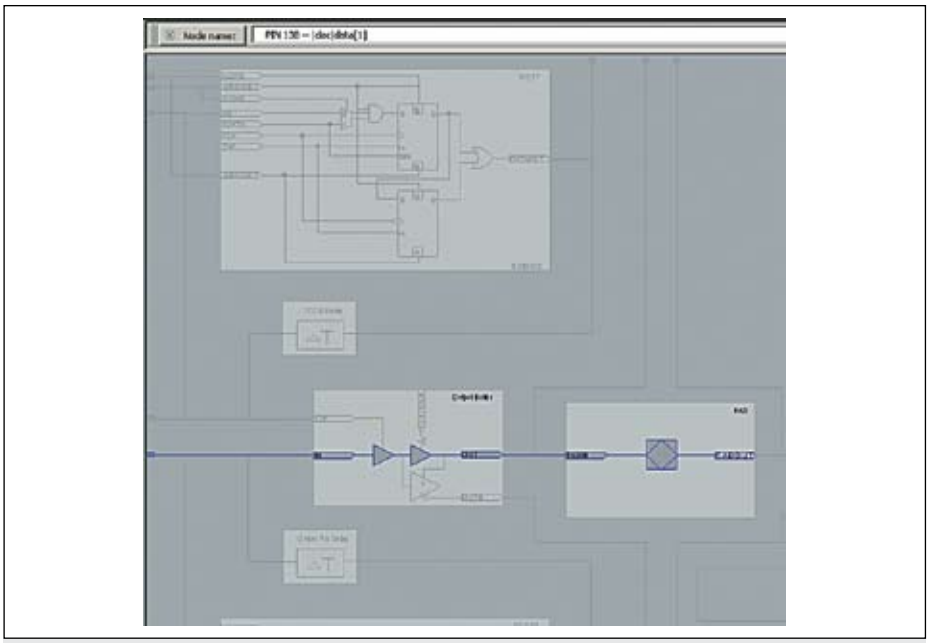


Рис. 28. Буфер ввода вывода data[1] проекта dac, в случае, когда триггера в нем нет

«Читать» подробные отчеты TimeQuest мы умеем (смотрите первую часть статьи). Поэтому не будем на этом останавливаться, а перейдем к сути временного нарушения. На рис. 25 видны задержки ***Clock Delay = 1.193ns*** и ***Data Delay = 3.318ns***. ***Clock Delay*** — это задержка от порта ПЛИС ***iclk*** до тактового входа триггера ***data[1]***. А ***Data Delay*** — это задержка от выхода триггера ***data[1]*** до порта ПЛИС ***data[1]***. Сумма этих величин дает ***Data Arrival***, то есть реальную задержку прибытия данных на ЦАП.

Теперь смотрим, что происходит с сигналом тактовой частоты ***oclk***. Видим ***Clock Delay = 2,051ns***. Это задержка сигнала ***oclk*** относительно сигнала ***iclk***. Прибавляем к этому времени требуемое нам ***th*** и получаем ***Data Required***, то есть требуемое время прибытия данных на порт ЦАП. Видно, что условие по ***th*** действительно не выполняется. Может быть, TimeQuest ошибается? Проверим задержку между портами ***iclk*** и ***oclk***. С помощью GUI выполним команду ***Report Path…*** (рис. 26). Действительно, не ошибается (рис. 27).

Имеем временное нарушение, которое нужно устранить. Первое, что приходит в голову: следует задержать данные относительно сигнала тактовой частоты, а задержка есть в буферах ввода/вывода ПЛИС. Посмотрим, используется ли она. С помощью команды ***Locate*** смотрим ***Resourse Property Editor*** (рис. 28). Занятно, а триггер ***data[1]*** вообще не находится в буфере ввода/вывода, а без этого триггера нельзя использовать управляемую задержку в ПЛИС семейства Cyclone III.

Назначаем свойство ***Fast Output Register*** на регистры ***data[*]***. Заново собираем проект, проверяем и видим, что временные ограничения по-прежнему не выполняются (рис. 29). Как же так? Проверим, использовал ли Quartus задержку в буфере ввода/вывода (рис. 30). Он сделал все, что мог, поставил триггер в буфер ввода/вывода и даже использовал задержку (***Output Pin Delay = 1***). Но ее не хватило

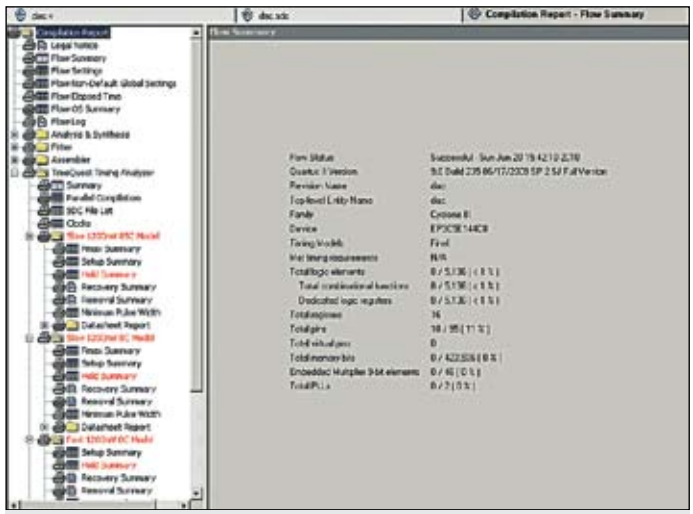


Рис. 29. Нарушения временных ограничений в проекте dac после размещения триггеров в буферах ввода/вывода

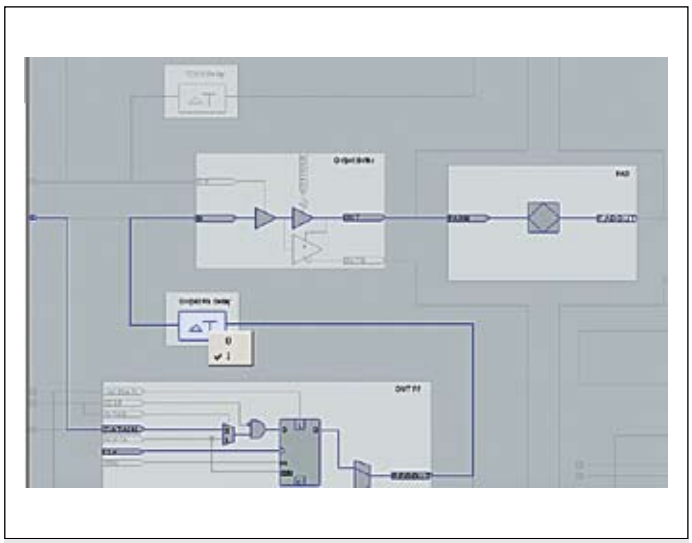


Рис. 30. Буфер ввода/вывода data[1] проекта dac в случае, когда триггер в нем есть

для выполнения временных ограничений. А функция подбора задержки за счет вставки цепочки LUT-ов в Quartus не реализована.

Как же быть? Тут можно вспомнить популярную, во времена устройств на дискретной логике, технику. Нужно подать на ЦАП инвертированный сигнал тактовой частоты. В этом случае при одинаковой задержке сигнала тактовой частоты и данных мы поставим фронт сигнала тактовой частоты в середину данных, что даст автоматическое выполнение временных ограничений по ***tsu/th***. Пишем в коде:

```
assign oclk = ~iclk;
```

И описываем это изменение в sdc-файле:

```
create_generated_clock -name {oclk} -invert -source [get_ports {iclk}]
[get_ports {oclk}]
```

Итак, ошибок нет (рис. 31) и еще больше 40 нс в запасе.

Автор хочет добавить, что метод инверсии сигнала тактовой частоты не является универсальным и использовать его везде и всегда не следует.

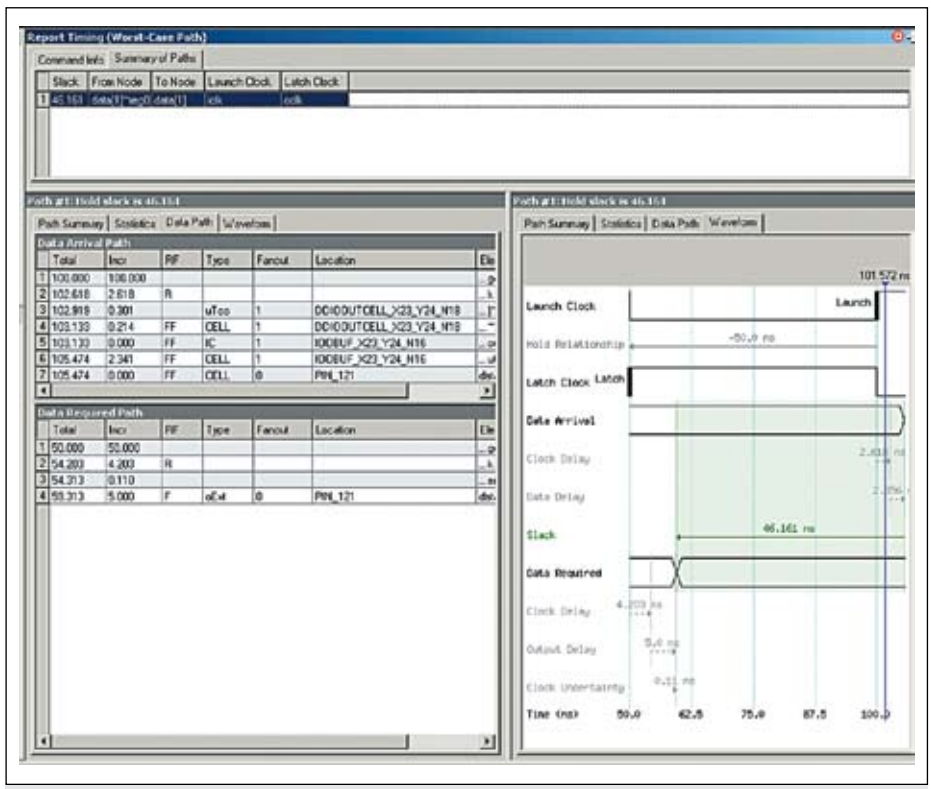


Рис. 31. Подробный отчет о временных ограничениях интерфейса проекта dac после инвертирования сигнала тактовой частоты ЦАП

Source-Synchronous Output

В интерфейсах этого типа (рис. 22) сигнал тактовой частоты идет на ПЛИС с отдельного генератора. Этот метод тактирования используется, когда требуется минимизировать джиттер тактового сигнала. В этом случае источник тактовой частоты для приемника интерфейсов вообще не присутствует в ПЛИС. Но мы помним, что для анализа синхронного интерфейса в TimeQuest требуется задать частоту и приемника, и передатчика. Как же быть? Для описания таких ситуаций в TimeQuest есть возможность задать так называемую виртуальную тактовую частоту, то есть тактовую частоту, у которой отсутствует физический источник.

Снова рассмотрим вывод пилы на наш ЦАП, на этот раз с внешним тактированием:

```
module dac (input iclk, output logic [7 : 0] data);

  logic [7 : 0] cnt;

  always_ff @(posedge iclk) begin
    cnt <= cnt + 1'b1;
    data <= cnt;
  end

endmodule
```

Соответствующий этому проекту sdc-файл будет следующий:

```
set_time_format -unit ns -decimal_places 3
derive_clock_uncertainty

create_clock -period 10MHz -name {iclk} [get_ports {iclk}]
create_clock -period 10MHz -name {virt_clk}

set_clock_groups -exclusive -group {iclk virt_clk}

# clock source to source clock pin delay
set clkBs_delay_max 0.3
set clkBs_delay_min 0.2
# clock source to destination clock pin delay
set clkBd_delay_max 1.1
set clkBd_delay_min 1.0
# source to destination data pins delay
set bdB_delay_max 0.5
set bdB_delay_min 0.4
# DAC parameters
set Tsu 5.0
set Th 5.0
```

```
set usedTsu [expr $clkBs_delay_max + $Tsu + $bdB_delay_max - $clkBd_delay_min]
set usedTh [expr $clkBd_delay_min - $Th + $bdB_delay_min - $clkBd_delay_max]

set_output_delay -clock [get_clocks {virt_clk}] -max $usedTsu [get_ports {data[*]}]
set_output_delay -clock [get_clocks {virt_clk}] -min $usedTh [get_ports {data[*]}]
```

На первый взгляд этот sdc-файл вызывает желание не связываться с такими интерфейсами и тактировать все от ПЛИС. Но не нужно торопиться: здесь все логично и просто. Начнем разбирать те строки sdc-файла, которые нам еще не знакомы.

ЦАП тактируется от того же генератора, что и ПЛИС, то есть в ПЛИС сигнала этой тактовой частоты физически не существует, но относительно ПЛИС он есть. Строка:

```
create_clock -period 10MHz -name {virt_clk}
```

описывает этот тактовый сигнал. Сами временные ограничения на ***tsu/th*** задаются аналогично интерфейсам System-Synchronous Output:

```
set_output_delay -clock [get_clocks {virt_clk}] -max $usedTsu [get_ports {data[*]}]
set_output_delay -clock [get_clocks {virt_clk}] -min $usedTh [get_ports {data[*]}]
```

Сложность заключается в определении значений TCL-переменных ***usedTsu*** и ***usedTh***. Задаются они следующим образом:

```
set usedTsu [expr $clkBs_delay_max + $Tsu + $bdB_delay_max - $clkBd_delay_min]
set usedTh [expr $clkBd_delay_min - $Th + $bdB_delay_min - $clkBd_delay_max]
```

Все TCL-переменные однозначно соответствуют именам путей на рис. 22. Для автоматического расчета значений мы используем возможность вычисления арифметических выражений в языке TCL. Это делается с помощью команды ***[expr]***, после которой можно использовать значения переменных и/или числовые значения. Для расчета временных ограничений потребовалось задать:

```
set clkBs_delay_max 0.3
set clkBs_delay_min 0.2
```

Это задержка тактового сигнала между генератором тактовой частоты и ПЛИС.

```
set clkBd_delay_max 1.1
set clkBd_delay_min 1.0
```

А это задержка тактового сигнала между генератором тактовой частоты и ЦАП. Эти задержки нужны для того, чтобы задать временной сдвиг этих сигналов относительно друг друга.

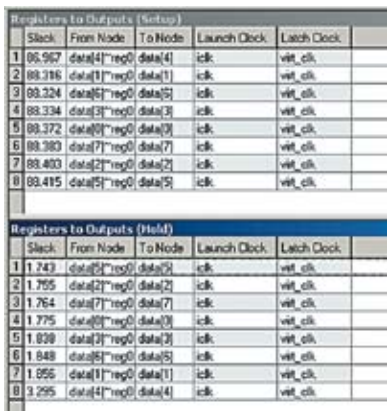
```
set bdB_delay_max 0.5
set bdB_delay_min 0.4
```

Это задержка сигналов данных между ЦАП и ПЛИС. Значения всех задаваемых задержек можно извлечь только из конструктива печатной платы. Задавать значения задержек «на авось» означает возможность получения нерабочего устройства.

```
set Tsu 5.0
set Th 5.0
```

Это уже знакомые нам временные параметры ЦАП. Для расчета ***usedTsu/usedTh*** используются такие значения задержек, которые, за счет сужения окна допустимых задержек, обеспечивают наихудшее значение запаса по ***tsu/th***.

Собираем проект, запускаем временной анализ, смотрим (рис. 32). Видим, что мы уложились в заданные ограничения, но посмотрите, какой большой запас есть по ***tsu*** и какой маленький запас по ***th***. Это не совсем хорошо, лучше всего, когда они сбалансированы между собой. Кроме того, в режиме временного анализа ***MIN_fast_1200mv_0c*** появляются ошибки по ***th*** (рис. 33).

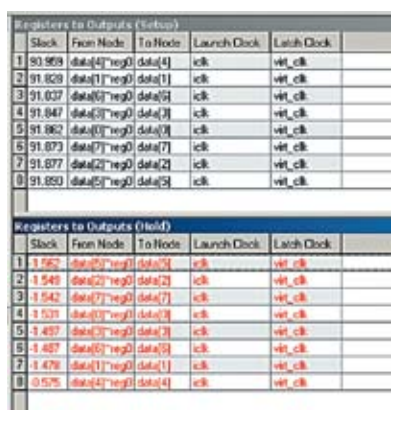

Registers to Outputs (Setup)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	86.967	data[4]"reg0	data[4]	clk	virt_clk
2	88.316	data[1]"reg0	data[1]	clk	virt_clk
3	88.324	data[5]"reg0	data[5]	clk	virt_clk
4	88.334	data[3]"reg0	data[3]	clk	virt_clk
5	88.372	data[0]"reg0	data[0]	clk	virt_clk
6	88.380	data[7]"reg0	data[7]	clk	virt_clk
7	88.403	data[2]"reg0	data[2]	clk	virt_clk
8	88.415	data[6]"reg0	data[6]	clk	virt_clk

Registers to Outputs (Hold)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	1.743	data[5]"reg0	data[5]	clk	virt_clk
2	1.755	data[2]"reg0	data[2]	clk	virt_clk
3	1.764	data[7]"reg0	data[7]	clk	virt_clk
4	1.775	data[0]"reg0	data[0]	clk	virt_clk
5	1.839	data[3]"reg0	data[3]	clk	virt_clk
6	1.848	data[6]"reg0	data[6]	clk	virt_clk
7	1.856	data[1]"reg0	data[1]	clk	virt_clk
8	3.295	data[4]"reg0	data[4]	clk	virt_clk

Рис. 32. Отчет о выполнении временных ограничений интерфейса с малым запасом по th


Registers to Outputs (Setup)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	90.969	data[4]"reg0	data[4]	clk	virt_clk
2	91.828	data[1]"reg0	data[1]	clk	virt_clk
3	91.837	data[6]"reg0	data[6]	clk	virt_clk
4	91.847	data[3]"reg0	data[3]	clk	virt_clk
5	91.862	data[0]"reg0	data[0]	clk	virt_clk
6	91.873	data[7]"reg0	data[7]	clk	virt_clk
7	91.877	data[2]"reg0	data[2]	clk	virt_clk
8	91.890	data[5]"reg0	data[5]	clk	virt_clk

Registers to Outputs (Hold)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	-1.567	data[5]"reg0	data[5]	clk	virt_clk
2	-1.548	data[2]"reg0	data[2]	clk	virt_clk
3	-1.542	data[7]"reg0	data[7]	clk	virt_clk
4	-1.531	data[0]"reg0	data[0]	clk	virt_clk
5	-1.457	data[3]"reg0	data[3]	clk	virt_clk
6	-1.457	data[6]"reg0	data[6]	clk	virt_clk
7	-1.478	data[1]"reg0	data[1]	clk	virt_clk
8	0.575	data[4]"reg0	data[4]	clk	virt_clk

Рис. 33. Отчет о выполнении временных ограничений интерфейса с малым запасом по th в режиме MIN_fast_1200mv_0c

Если посмотреть внимательнее (рис. 34), то видно, что нам не хватает задержки по данным. А Quartus не умеет набирать эту задержку на LUT-ах. В данном примере ***Fast Output Register*** используется, но все равно задержки не хватает.

В таком случае более эффективна техника многофазной синхронизации, описанная в первой части статьи, а именно с помощью PLL подвинуть тактовую частоту ПЛИС. С помощью MegaWizard генерируем PLL, сдвигающую сигнал тактовой частоты по фазе на 90°:

```
module dac (input iclk, output logic [7 : 0] data);

   pll pll (iclk, used_clk);

   logic [7 : 0] cnt;

   always_ff @(posedge used_clk) begin
      cnt <= cnt + 1'b1;
      data <= cnt;
   end

endmodule
```

Ошибок нет (рис. 35), и видно перераспределение запаса между ***tsu/th***. Запас изменился на ~25 нс, как и должно быть при сдвиге фазы частоты 10 МГц на 90°.

Мы рассмотрели тактирование источника данных и ПЛИС от одного генератора. В случае если ПЛИС тактируется от источника данных (например, от шины процессора в режиме Master), то в формуле расчета требуемых задержек нужно положить задержку сигнала тактовой частоты до источника данных равной нулю, то есть:

```
# clock source to source clock pin delay
set clkBs_delay_max 0.0
set clkBs_delay_min 0.0
```

На этом мы закончим рассмотрение выходных интерфейсов и перейдем к входным. Отметим тот факт, что если в выходных интерфейсах мы задаем временные ограничения исходя из условия выполнения ***tsu/th*** триггеров вне ПЛИС, то во входных интерфейсах — исходя из условия выполнения ***tsu/th*** триггеров в ПЛИС.

System-Synchronous Input

Рассмотрим тактирование АЦП от ПЛИС. В качестве примера возьмем параллельный АЦП AD9215. Предположим, что частота тактирования составляет 50 МГц:

```
module adc (input clk, input [9 : 0] adc_dat, output logic adc_clk,
output logic [9 : 0] data);

   logic [9 : 0] adc_io_reg;
   logic [9 : 0] adc_reg;

   always_ff @(posedge clk) begin
      {adc_reg, adc_io_reg} <= {adc_io_reg, adc_dat};
   end

   assign adc_clk = clk;

   always_ff @(posedge clk) begin
      data <= adc_reg;
   end

endmodule
```

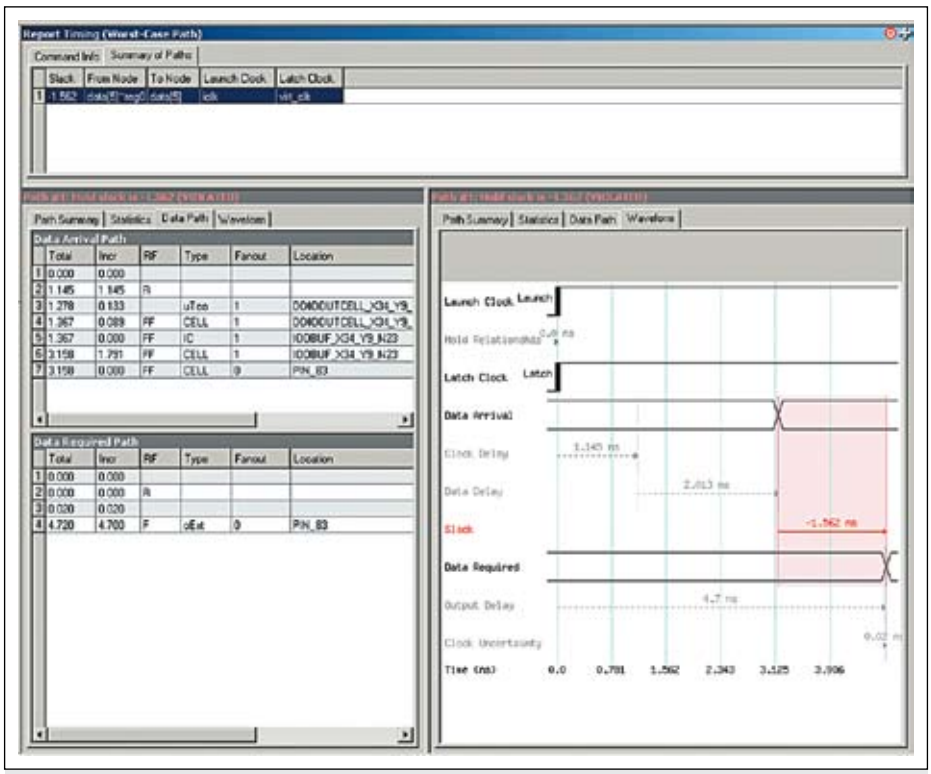


Рис. 34. Подробный отчет о временных ограничениях интерфейса с малым запасом по th

Sdc-файл для этого проекта будет такой:

```
set_time_format -unit ns -decimal_places 3

derive_clock_uncertainty

create_clock -period 50MHz -name {clk} [get_ports {clk}]
create_generated_clock -name {adc_clk} -source [get_ports {clk}]
[get_ports {adc_clk}]
#clock source to destination clock pin delay
set clkA_delay_max [expr 30.0*0.010]
set clkA_delay_min [expr 30.0*0.005]
# source to destination data pins delay
set bdA_delay_max [expr 30.0*0.010]
set bdA_delay_min [expr 30.0*0.005]
#ADC parameters
set Tco_max 6.5
set Tco_min 2.5

set usedTsu [expr $clkA_delay_max + $Tco_max + $bdA_delay_max]
set usedTh [expr $clkA_delay_min + $Tco_min + $bdA_delay_min]

set_input_delay -clock {adc_clk} -max $usedTsu [get_ports {adc_dat[*]}]
set_input_delay -clock {adc_clk} -min $usedTh [get_ports {adc_dat[*]}]
```

Файл задания ограничений для этого проекта в части задания частот ПЛИС и АЦП похож на файл для проекта с System-Synchronous Output. Помимо тактовых частот, потребовалось задать:

```
set clkA_delay_max [expr 30.0*0.010]
set clkA_delay_min [expr 30.0*0.005]
```

Это задержка сигнала тактовой частоты от ПЛИС до АЦП (рис. 21).

```
set bdA_delay_max [expr 30.0*0.010]
set bdA_delay_min [expr 30.0*0.005]
```

А это задержка сигнала данных от АЦП к ПЛИС. Предположим, что на плате это проводник длиной 30 мм. 0,007 нс/мм — это оценочная задержка проводника шириной 0,2 мм на текстолите марки FR4. Для усложнения анализа мы задаем разброс этой задержки. Сам расчет, с помощью уже знакомой нам команды ***[expr]***, поручаем TimeQuest.

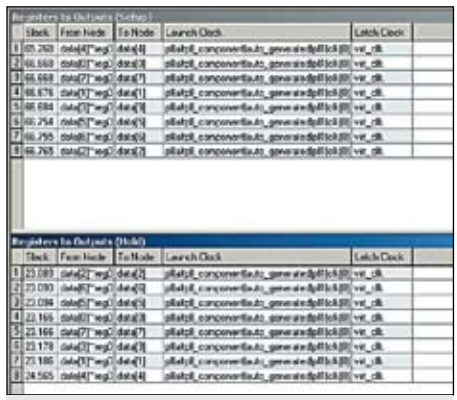


Рис. 35. Подробный отчет о временных ограничениях интерфейса после сдвига тактового сигнала данных на 90°

```
set Tco_max 6.5
set Tco_min 2.5
```

Параметры АЦП возьмем из документации. Так как для анализа нужно получить оценку для наихудшего случая, то вместо типовых значений используются крайние значения ***Tco*** АЦП.

Временные ограничения для выполнения ***tsu/th*** во входных триггерах ПЛИС описываются так:

```
set usedTsu [expr $clkA_delay_max + $Tco_max + $bdA_delay_max]
set usedTh [expr $clkA_delay_min + $Tco_min + $bdA_delay_min]

set_input_delay -clock {adc_clk} -max $usedTsu [get_ports {adc_dat[*]}]
set_input_delay -clock {adc_clk} -min $usedTh [get_ports {adc_dat[*]}]
```

Видно, что в эти ограничения входят задержки сигналов и параметры АЦП, соответствующие наихудшему случаю. Собираем, запускаем, смотрим (рис. 36). Видим, что все

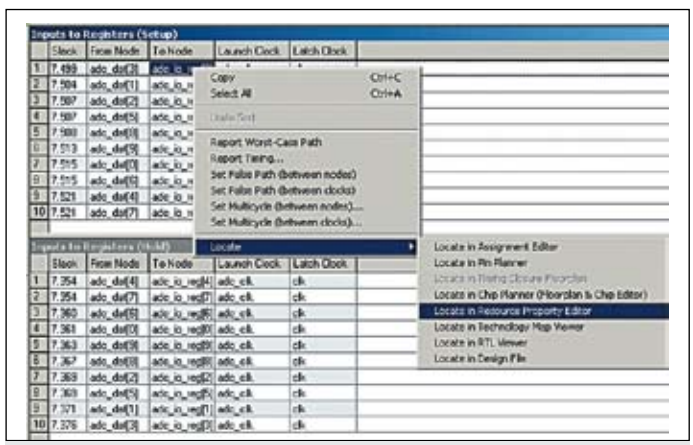


Рис. 36. Отчет о временных ограничениях интерфейса проекта adc для System-Synchronus Input

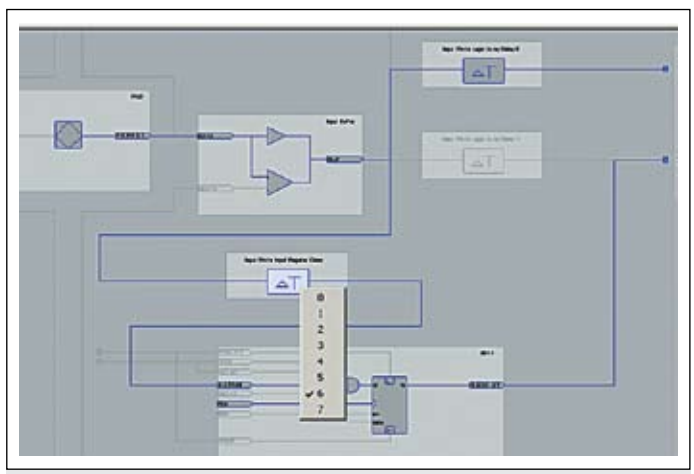
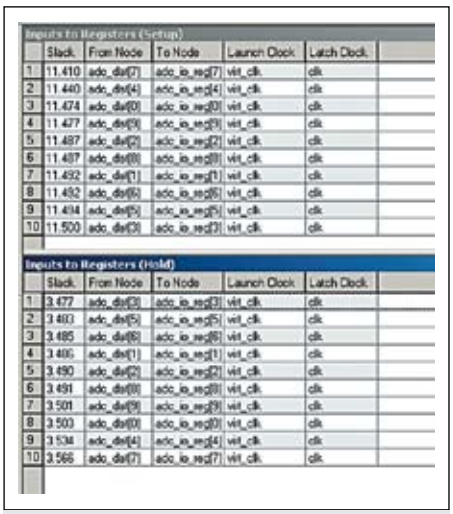


Рис. 37. Буфер ввода/вывода проекта adc для System-Synchronus Input



Inputs to Registers (Setup)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	11.410	adc_dat[7]	adc_io_reg[7]	virt_clk	clk
2	11.440	adc_dat[4]	adc_io_reg[4]	virt_clk	clk
3	11.474	adc_dat[0]	adc_io_reg[0]	virt_clk	clk
4	11.477	adc_dat[9]	adc_io_reg[9]	virt_clk	clk
5	11.487	adc_dat[2]	adc_io_reg[2]	virt_clk	clk
6	11.487	adc_dat[8]	adc_io_reg[8]	virt_clk	clk
7	11.492	adc_dat[1]	adc_io_reg[1]	virt_clk	clk
8	11.492	adc_dat[6]	adc_io_reg[6]	virt_clk	clk
9	11.494	adc_dat[5]	adc_io_reg[5]	virt_clk	clk
10	11.500	adc_dat[3]	adc_io_reg[3]	virt_clk	clk

Inputs to Registers (Hold)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	3.477	adc_dat[3]	adc_io_reg[3]	virt_clk	clk
2	3.480	adc_dat[5]	adc_io_reg[5]	virt_clk	clk
3	3.485	adc_dat[6]	adc_io_reg[6]	virt_clk	clk
4	3.486	adc_dat[1]	adc_io_reg[1]	virt_clk	clk
5	3.490	adc_dat[2]	adc_io_reg[2]	virt_clk	clk
6	3.491	adc_dat[8]	adc_io_reg[8]	virt_clk	clk
7	3.501	adc_dat[9]	adc_io_reg[9]	virt_clk	clk
8	3.503	adc_dat[0]	adc_io_reg[0]	virt_clk	clk
9	3.534	adc_dat[4]	adc_io_reg[4]	virt_clk	clk
10	3.566	adc_dat[7]	adc_io_reg[7]	virt_clk	clk

Рис. 38. Отчет о временных ограничениях интерфейса проекта adc для Source-Synchronous Input

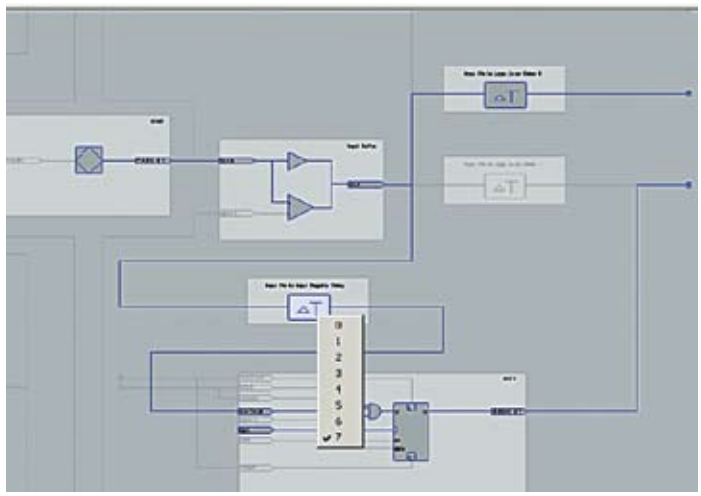


Рис. 39. Буфер ввода/вывода проекта adc для Source-Synchronous Input

заданные ограничения выполнены и запас по ***tsu/th*** распределен равномерно. За счет чего Quartus смог добиться такого хорошего результата? С помощью команды ***Locate*** в ***Resource Property Editor*** (рис. 36) смотрим, что сделал Quartus в ячейке ввода/вывода (рис. 37). Он автоматически подобрал такое значение модуля управляемой задержки, которое позволило оптимально сбалансировать запас. У кого-то еще остались сомнения в нужности задания временных ограничений и в использовании TimeQuest?

Source-Synchronous Input

Вот мы и подошли к последнему из рассматриваемых нами синхронных интерфейсов. Рассмотрим тот же самый АЦП, но тактируемый от отдельного генератора:

```
module adc (input clk, input [9 : 0] adc_dat, output logic [9 : 0] data);

   logic [9 : 0] adc_io_reg;
   logic [9 : 0] adc_reg;

   always_ff @(posedge clk) begin
      {adc_reg, adc_io_reg} <= {adc_io_reg, adc_dat};
   end

   always_ff @(posedge clk) begin
      data <= adc_reg;
   end

endmodule
```

Как и при Source-Synchronous Output, sdc-файл для этого проекта будет содержать описание виртуального сигнала тактовой частоты вместе с сигналом тактовой частоты, помещенного в одну логическую группу:

```
set_time_format -unit ns -decimal_places 3

derive_clock_uncertainty

create_clock -period 50MHz -name {clk} [get_ports {clk}]
create_clock -period 50MHz -name {virt_clk}

set_clock_groups -exclusive -group {clk virt_clk}

# clock source to source clock pin delay
set clkAs_delay_max [expr 30.0*0.010]
set clkAs_delay_min [expr 30.0*0.005]
# clock source to destination clock pin delay
set clkAd_delay_max [expr 30.0*0.010]
set clkAd_delay_min [expr 30.0*0.005]
# source to destination data pins delay
set bdA_delay_max [expr 30.0*0.010]
set bdA_delay_min [expr 30.0*0.005]
#ADC parameters
set Tco_max 6.5
set Tco_min 2.5

set usedTsu [expr $clkAs_delay_max + $Tco_max + $bdA_delay_max
- $clkAd_delay_min]
set usedTh [expr $clkAs_delay_min + $Tco_min + $bdA_delay_min
- $clkAd_delay_max]

set_input_delay -clock {virt_clk} -max $usedTsu [get_ports {adc_dat[*]}]
set_input_delay -clock {virt_clk} -min $usedTh [get_ports {adc_dat[*]}]
```

И в отличие от System-Synchronous Input, во временных ограничениях будут учитываться задержки от генератора тактовой частоты до ПЛИС и АЦП:

```
set usedTsu [expr $clkAs_delay_max + $Tco_max + $bdA_delay_max
- $clkAd_delay_min]
set usedTh [expr $clkAs_delay_min + $Tco_min + $bdA_delay_min
- $clkAd_delay_max]
```

Собираем, запускаем, смотрим (рис. 38). Временные ограничения выполняются, но запас по ***tsu/th*** распределен неравномерно. Для выяснения причин такой неравномерности опять обратимся к Resource Property Editor (рис. 39). Quartus использовал максимальную задержку, чтобы увеличить запас по ***th***, но ее хватило только на такой запас.

Можно оставить в проекте все как есть, так как запаса в 3,4 нс для работы хватит. А можно воспользоваться PLL и подвинуть фазу частоты тактирования триггеров. В качестве домашнего задания это предлагается сделать самостоятельно.

На этом мы закончили рассмотрение синхронных интерфейсов. Для закрепления материала автор рекомендует взять рассмотренные проекты и, изменяя разные временные параметры в sdc-файлах, смотреть, как они влияют на результат анализа.

Асинхронные интерфейсы

Асинхронные интерфейсы подразделяются на асинхронные с обработкой на комбинационной логике и асинхронные с обработкой на системной тактовой частоте. Способ наложения временных ограничений на эти интерфейсы сильно зависит от вида интерфейсов. Рассмотрим типовые случаи.

Асинхронные интерфейсы с обработкой на комбинационной логике

К этой категории интерфейсов относятся различные дешифраторы, стыковая логика, защелки расширителей сигналов и т. д. В большинстве проектов можно без какого-либо ухудшения качества реализации проекта объявить все цепи как пути, которые не надо анализировать (***set_false_path***). В любом случае задержки там будут не более 20–40 нс, что для большинства применений более чем достаточно. Это связано с тем, что Quartus при синтезе и разводке старается минимизировать задержку всех цепей.

Но в практике проектирования для ПЛИС есть случаи, когда требуется точное задание различных временных ограничений. К этим случаям относятся проекты, содержащие логику триггеров-защелок (latch), стробируемые тактовые частоты (gated clock) или закладывающие в свою функциональность использование определенного значения задержки. Подобные методы описания часто встречаются у начинающих разработчиков (например, в проектах подают комбинационные сигналы на тактовые входы триггеров). Часто удивляются, почему не работает. На сленге такая техника называется «асинхронщиной», и ее стараются избегать. Давайте рассмотрим почему.

Возьмем простой асинхронный декодер:

```
module async_decoder (input [1 : 0] idat, output logic odat);
   assign odat = &idat;
endmodule
```

Для того чтобы сигнал ***odat*** во время переключений сигналов ***idat*** содержал выбросы (glitch) минимальной длительности, нужно выровнять задержку сигналов до него. То есть нас интересует относительный перекос задержки сигналов ***idat[0]*** и ***idat[1]*** до сигнала ***odat***. В TimeQuest временные ограничения для задержек асинхронных путей могут быть заданы с помощью следующих команд:

- Команд задающих абсолютную задержку цепи между источником и приемником сигнала:

```
set_net_delay -from <names> [-max] [-min] -to <names> <delay>

set_max_delay -from <names> -to <names> <delay>
set_min_delay -from <names> -to <names> <delay>
```

- Команды задающей максимальный перекос задержек цепей между источниками и приемниками сигнала:

```
set_max_skew -from <names> -to <names> <skew>
```

Казалось бы, последняя команда — это то, что нам нужно. Но детальное изучение документации на Quartus приводит к неутешительному выводу:

```
The Fitter does not include set_max_skew constraints in design optimization. Use placement, routing, or other timing constraints to drive the fitter to meet any set_max_skew constraints.
```

То есть перекос сигналов можно задать и измерить (***report_max_skew***), но на него нельзя повлиять. Эта команда носит исключительно демонстрационный характер.

Остается только задавать допустимый диапазон абсолютных задержек. Воспользуемся командой ***set_net_delay***. Тут нас ждет первый сюрприз. Особенность этой команды в том, что в качестве источников/приемников могут использоваться только имена пинов логических элементов ПЛИС. Поэтому вместо краткой и понятной записи:

```
set_net_delay -from [get_ports {idat[*]}] -max -to [get_ports {odat}] 5.2
set_net_delay -from [get_ports {idat[*]}] -min -to [get_ports {odat}] 5.1
```

придется использовать такую:

```
set_net_delay -from [get_ports {idat[*]}] -max 0.2 -to [get_pins {idat[*]~inputlo}]
set_net_delay -from [get_ports {idat[*]}] -min 0.1 -to [get_pins {idat[*]~inputlo}]

set_net_delay -from [get_pins {idat[*]~inputlo}] -max 0.4 -to [get_pins {WideAnd0|combout}]
set_net_delay -from [get_pins {idat[*]~inputlo}] -min 0.3 -to [get_pins {WideAnd0|combout}]

set_net_delay -from [get_pins {WideAnd0|combout}] -max 0.6 -to [get_pins {odat~outputlo}]
set_net_delay -from [get_pins {WideAnd0|combout}] -min 0.5 -to [get_pins {odat~outputlo}]

set_net_delay -from [get_pins {odat~outputlo}] -max 0.8 -to [get_ports {odat}]
set_net_delay -from [get_pins {odat~outputlo}] -min 0.7 -to [get_ports {odat}]
```

То есть фактически придется переписать результат работы синтезатора и разводчика

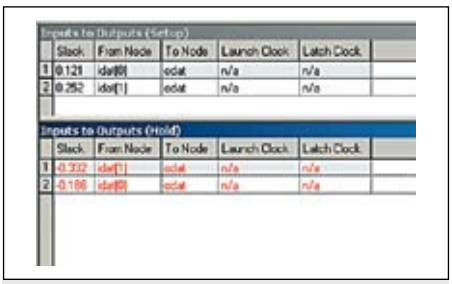


Рис. 40. Отчет о временных нарушениях для проекта async_decoder в режиме 8_slow_1200mv_85c

вручную. Так придется поступать каждый раз при смене семейства ПЛИС и/или изменении проекта. Но на этом сюрпризы не закончились. Оказывается, что TimeQuest даже не считает такое задание задержки цепи критическим ограничением. На выполнение команды ***Report Top Falling Paths*** он утверждает, что все в порядке. Посмотреть отчет о задержках, заданных через ***set_net_delay***, можно только с помощью отдельной команды ***report_net_delay***. После таких сюрпризов тот факт, что Quartus не может самостоятельно выравнивать задержки, заданные таким образом, уже не удивляет.

Остается последний вариант задания задержек:

```
set_max_delay -from [get_ports {idat[*]}] -to [get_ports {odat} ] 5.2
set_min_delay -from [get_ports {idat[*]}] -to [get_ports {odat} ] 5.1
```

Собираем проект, запускаем анализ и видим интересный результат (рис. 40, 41).

Задержка сильно зависит от температуры и от конкретной микросхемы ПЛИС. Поэтому точно и однозначно задать задержку цепи нельзя. Именно поэтому использование асинхронной логики в проектах на ПЛИС не рекомендуется, так как очень сложно получить гарантированно рабочий вариант. Автор не утверждает, что асинхронный дизайн использовать нельзя, можно, но отдавайте себе отчет в том, что вы делаете.

Если все так плохо, то как же тогда поступить в случае необходимости реализации в ПЛИС асинхронной защелки? Рассмотрим этот пример:

```
module async_decoder (input ce, input [1 : 0] idat, output logic odat);
   always_ff @(posedge ce) odat = &idat;
endmodule
```

Для задания временных ограничений такого проекта автор предлагает воспользоваться методами, используемыми для синхронных интерфейсов. А именно использовать следующий sdc-файл:

```
create_clock -name ce -period 100MHz [get_ports {ce}]
set_input_delay -clock [get_clocks {ce}] -max 5.1 [get_ports {idat[*]}]
set_input_delay -clock [get_clocks {ce}] -min 5.0 [get_ports {idat[*]}]
```

То есть объявить сигнал ***ce*** как тактовую частоту и задать задержки относительно его.

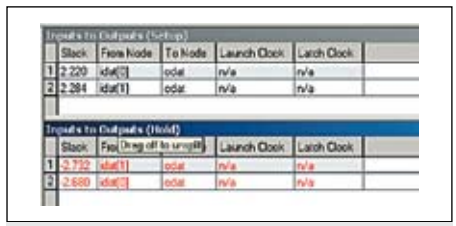


Рис. 41. Отчет о временных нарушениях для проекта async_decoder в режиме MIN_fast_1200mv_0c

Временной анализ этого проекта предлагается выполнить читателю самостоятельно, в качестве домашнего задания.

Асинхронные интерфейсы с обработкой на системной тактовой частоте

В качестве яркого примера подобного асинхронного интерфейса рассмотрим всем известный, самый обычный UART. При реализации этого интерфейса приемный сигнал UART_TX дискретизируют системной частотой, которая должна быть выше частоты UART, и обрабатывают, используя детекторы перехода сигнала из состояния в состояние.

Временные ограничения для таких интерфейсов — это знакомые нам ложные пути (***false_path***):

```
set_false_path -from [get_ports {uart_rx}] -to [get_clocks{sys_clk}]
```

Так мы задаем путь, который не нужно анализировать, от порта ***uart_rx*** до триггеров, тактируемых частотой ***sys_clk***.

```
set_false_path -from [get_ports {uart_rx}] -to [all_clocks]
```

А так — путь от порта ***uart_rx*** до триггеров, тактируемых от **любой** тактовой частоты в системе. С выходным сигналом поступаем точно так же:

```
set_false_path -from [all_clocks] -to [get_ports {uart_tx}]
```

Стоит отметить, что в данном случае (UART) можно даже не контролировать размещение триггеров ***uart_rx/uart_tx*** в буфере ввода/вывода ПЛИС.

Рассмотрим более сложный пример, возьмем простой SPI-мастер. Положим, что данные захватываются слейвом по фронту тактовой частоты:

```
module spi (input clk, start, input [7 : 0] data, output logic busy, sclk, sdi, cs_n) ;

   logic ff ;
   logic [3 : 0] cnt ;
   logic done;
   logic [7 : 0] buffer;

   assign ff = cnt[0];
   assign done = &cnt;

   always_ff @(posedge clk) begin // simple FSM
      if (~busy) begin
```

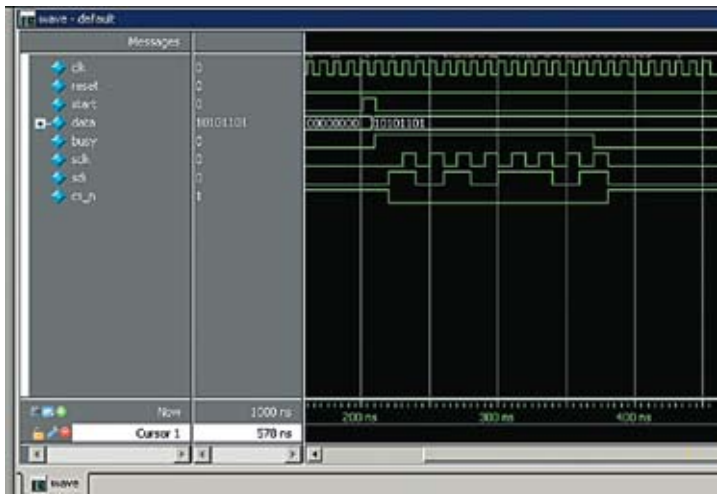


Рис. 42. Временные диаграммы работы проекта spi

```
        if (start) begin
            busy <= 1'b1;
            cnt <= 0;
            buffer <= data;
        end
    end
    else begin
        cnt <= cnt + 1'b1;
        if (ff) begin
            buffer <= (buffer << 1);
        end
        if (done) begin
            busy <= 1'b0;
        end
    end
  end

  always_ff @(posedge clk) begin // io registers
    sclk <= ff;
    sdi <= buffer[7];
    cs_n <= ~busy;
  end

endmodule
```

SPI — это синхронный интерфейс, так как вместе с данными передается сигнал тактовой частоты ***sclk***. Но, судя по временным диаграммам (рис. 42), временные соотношения между сигналами интерфейса заданы последовательностью состояний конечного автомата.

Единственное, что нужно для их четкого выполнения, это убедиться в том, что выходные триггеры были размещены в ячейках ввода/вывода ПЛИС. Как вы уже, наверное, догадались, временные ограничения для этого интерфейса будут:

```
set_false_path -from [get_clocks {clk}] -to [get_ports {sclk sdi cs_n}]
```

Заключение

Мы закончили рассмотрение самой большой и требующей ясного понимания области задания временных ограничений. Все формулы вычисления значений тех или иных параметров, приведенные в этой части статьи, логически обоснованы. Достаточно понять физический смысл этих параметров, и задание временных ограничений для того или иного интерфейса будет простой задачей.

В последней части статьи мы рассмотрим основы использования исключений задания временных ограничений. К ним относятся мультицикловые и ложные пути (***multicycle path/false_path***). Для эффективной работы в Quartus и TimeQuest эту область нужно обязательно освоить. Вы увидите, что и это не представляет большой сложности.

Литература

1. Quartus II Handbook Version 9.0 — **http://www.altera.com/literature/lit-qts.jsp**
2. SDC and TimeQuest API Reference Manual — **http://www.altera.com/literature/manual/mnl_sdctmq.pdf**
3. Quartus II TimeQuest Timing Analyzer Cookbook — **http://www.altera.com/literature/manual/mnl_timequest_cookbook.pdf**
4. Constraining and Analyzing Source-Synchronous Interfaces — **http://www.altera.com/literature/an/an433.pdf**
5. Шехалев Д. Synopsys Design Constraint — язык задания временных ограничений на примере Altera Time Quest. Часть 1 и часть 2 // Компоненты и технологии. 2010. № 9, 10.
6. **http://embedders.org/blog/des00**